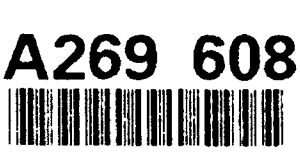


12

Bias in Planning and Explanation-Based Learning

Paul S. Rosenbloon, Soowon Lee and Amy Unruh
USC/ Information Sciences Institute
4676 Admiralty Way
Marina del Rey, California 90292

May 1993
ISI/RR-93-346

REPORT DOCUMENTATION PAGE

FORM APPROVED
OMB NO. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	May 1993	Research Report

4. TITLE AND SUBTITLE
Bias in Planning and Explanation-Based Learning

5. FUNDING NUMBERS
N00014-89-K-0155

6. AUTHOR(S)
Paul S. Rosenbloom, Soowon Lee and Amy Unruh

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
USC INFORMATION SCIENCES INSTITUTE
4676 ADMIRALTY WAY
MARINA DEL REY, CA 90292-6695

8. PERFORMING ORGANIZATON REPORT NUMBER
RR-346

9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES)
ARPA
3701 Fairfax Drive
Arlington, VA 22203

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12A. DISTRIBUTION/AVAILABILITY STATEMENT
UNCLASSIFIED/UNLIMITED

12B. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 words)***

Biases enable systems to make decisions in realms where all legitimate sources of knowledge have been exhausted. This article investigates the application of biases to the problem of planning, and how this can indirectly induce effective biases in a learning process that is based on planner's experiences. Experimental results from six biased planners, plus several more complex multi-method planners, indicate complex tradeoffs among planner completeness, planning efficiency, and length. Learning also varies in complex ways among these planners, with one notable result being the ease with which some planners learn rules that can generalize from one object to many; a phenomenon known in machine learning as *generalization to N*.

14. SUBJECT TERMS
Bias, Planning, Explanation-based learning, Multi-method planners, Soar

15. NUMBER OF PAGES
39

16. PRICE CODE

17. SECURITY CLASSIFICTION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UNLIMITED

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reoprts. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to stay within the lines to meet optical scanning requirements.

Block 1. Agency Use Only (Leave blank).

Block 2. Report Date. Full publication date including day, month,a nd year, if available (e.g. 1 jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element numbers(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C - Contract	PR - Project
G - Grant	TA - Task
PE - Program Element	WU - Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the repor.-

Block 9. Sponsoring/Monitoring Agency Names(s) and Address(es). Self-explanatory

Block 10. Sponsoring/Monitoring Agency Report Number. (If known)

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of ...; To be published in... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

DOD	- See DoDD 5230.24, "Distribution Statements on Technical Documents."
DOE	- See authorities.
NASA	- See Handbook NHB 2200.2.
NTIS	- Leave blank.

Block 12b. Distribution Code.

DOD	- Leave blank.
DOE	- Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.
NASA	- Leave blank.
NTIS	- Leave blank.

Block 13. Abstract. Include a brief (Maximum 200 words) factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code (NTIS only).

Blocks 17.-19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contins classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.

Bias in Planning and Explanation-Based Learning[1]

Paul S. Rosenbloom, Soowon Lee
Information Sciences Institute and
Computer Science Department
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292

Amy Unruh
Knowledge Systems Laboratory
Computer Science Department
Stanford University
701 Welch Rd. (Bldg. C)
Stanford, CA 94305

To appear in S. Chipman & A. Meyrowitz (Eds.), *Foundations of Knowledge Acquisition: Cognitive Models of Complex Learning*. Hingham, MA: Kluwer Academic Publishers. 1993.

Abstract

Biases enable systems to make decisions in realms where all legitimate sources of knowledge have been exhausted. This article investigates the application of biases to the problem of planning, and how this can indirectly induce effective biases in a learning process that is based on a planner's experiences. Experimental results from six biased planners, plus several more complex multi-method planners, indicate complex tradeoffs among planner completeness, planning efficiency, and plan length. Learning also varies in complex ways among these planners, with one notable result being the ease with which some planners learn rules that can generalize from one object to many; a phenomenon known in machine learning as *generalization to N*.

1. We would like to thank Mike Barley, Pat Langley, and Steve Minton for their helpful feedback on this work. This work was sponsored by the Defense Advanced Research Projects Agency (DOD) and the Office of Naval Research under contract number N00014-89-K-0155. This chapter is a reformatted reprint of a version of Rosenbloom, P. S., Lee, S. & Unruh, A. (1992). Bias in planning and explanation-based learning. In S. Minton (Ed.), *Machine Learning Methods for Planning and Scheduling*. San Mateo, CA: Morgan Kaufmann. In Press.

INTRODUCTION

Bias, as originally defined in the context of inductive concept learning from preclassified training instances, is "any basis for choosing one generalization over another, other than strict consistency with the observed training instances (Mitchell, 1980)." It has proven to be a particularly useful notion in this context because it isolates and highlights a crucial aspect of induction algorithms: the knowledge and processes that determine how the algorithms go beyond the training instances; that is, which inductive leaps they make. For example, it makes it clear that explanation-based learning (EBL) can be viewed as inductive concept learning, where the domain theory and operationality criterion provide a particularly strong bias on the induction process. In general, the idea in induction is to start with the notion of an *unbiased hypothesis space* that consists of every possible generalization of the observed training instances. The *unbiased version space* is then the portion of this space that is consistent with the observed training instances. The bias determines which element — if any — of the unbiased version space is returned as the output of the induction algorithm.[2]

As just described, bias affects the output of the induction process, but not the efficiency with which it proceeds. This is because bias is implicitly used solely as part of the test for a generate-and-test method — first the elements of the unbiased hypothesis space are generated, and then tested to see if they meet the criteria of the bias. However, if the bias can be incorporated directly into the generator (Bennett & Dietterich, 1986), then it can also have a significant impact on the efficiency of the induction process by reducing the number of candidates that are generated. In this way, bias can lead to effective control of search. However, despite this close relationship between search control and bias — as we shall see later, search control can also lead to bias — the two notions are not isomorphic. Bias determines which answer is given, while search control determines the efficiency with which that answer is found.

The principal thesis of this chapter is that the notion of a bias — suitably generalized — can also be usefully applied to planning. Several

2. In search terms, the set of possible generalizations provides a problem (or solution) space, consistency with the observed training instances provides a goal test, and the bias determines which of the states that satisfy the goal is actually reached.

of the potential benefits are straightforward mappings from the inductive concept learning case: (1) it can help to organize and understand many of the concepts in planning -- such as linearity and protection -- by focusing on their effects on selection from the hypothesis space (that is, the *plan space*); and (2) it can reduce computational effort by reducing the number of hypotheses (plans) that must be examined if a good bias is selected.

A third potential benefit, and one that is not derived from the standard usage of bias in inductive concept learning, is that planning biases can indirectly induce an effective bias on learning. The basic issue here is how to make the rules learned from planning episodes more utile than they would be otherwise. Without such modifications, the rules may actually hurt performance rather than help it (Minton, 1990; Tambe, Newell, & Rosenbloom, 1990). In fact, much of the research in plan learning over the past several years can be construed as investigating the direct application of biases to improve the utility of learned rules; for example, ULS (Chase et al., 1989) uses statistical information to abstract learned rules by dropping conditions that have a high conditional probability of being true given that the preceding conditions are also true. The main problem with this type of approach is that it uses the biasing knowledge only for post hoc revision of the learned rules, and not to assist the planner in doing its job. Thus an opportunity is missed to reduce planning effort, and thus to reduce learning time, since learning time is closely linked to the time required by the planner for it to reach situations where a rule can be learned. The alternative approach is to bias the planner — for example, by having it create plans that ignore preconditions of little statistical relevance -- and then to learn from these altered planning episodes. Recent evidence shows that, at least for some forms of abstraction, this approach can reduce planning time (and thus reduce learning time), and increase the generality and utility of the rules learned (Unruh & Rosenbloom, 1989; Knoblock, Minton, & Etzioni, 1991).

In the following three sections of this chapter we lay out in more detail the application of bias to planning, describe how (biased) planning can be implemented within Soar (Laird, Newell, & Rosenbloom, 1987; Rosenbloom, Laird, & Newell, in press), and provide results from some of the biases implemented so far. This is followed with an investigation of one

approach to the flexible use of multiple biases within a single planner, by constructing a set of multi-method planners out of sequences of increasingly less biased planners. Such multi-method planners can alter the trade-offs among planning efficiency, plan length, and planner completeness. They also provide an opportunity to investigate how to learn about which planners — and thus which biases — to use for particular problems. The remainder of the chapter examines whether biased planning can lead to useful biases on learning; in particular, on explanation-based learning (EBL) of plans. This proceeds via a case study in generalization to N (Boström, 1990; Cohen, 1988; Shavlik, 1989; Subramanian & Feldman, 1990).

BIAS IN PLANNING

Figure 1 displays the analogy between inductive concept learning and planning that underlies the transfer of the notion of bias to planning. In both cases the output of the process is to be some element of the unbiased hypothesis space that is consistent with the process's input. Where the two cases differ is in the definitions of "unbiased hypothesis space" and "input". In concept learning, the unbiased hypothesis space is the power set of the possible instances, and the input is a set of preclassified training instances. In planning, the unbiased hypothesis space is the power sequence — that is, the set of all sequences — of the possible operators,[3] and the input is the combination of an initial state and a goal. In either case — despite these differences — in the absence of a bias, any element of the hypothesis space consistent with the input (that is, any element of the version space) is as good as any other. Thus, in both cases, it is the bias that breaks this deadlock and determines which such element becomes the output of the process.

Biases can be either absolute or relative. An absolute bias completely removes regions of the unbiased hypothesis space, creating an incomplete *biased hypothesis space*. For example, in concept learning, a generalization language provides an absolute bias by eliminating any element of the

3. The specification here assumes that the plan space contains only totally-ordered sequences of operators, but it does not rule out a search strategy that incrementally specifies an element of the plan space by refining a partially-ordered plan structure.

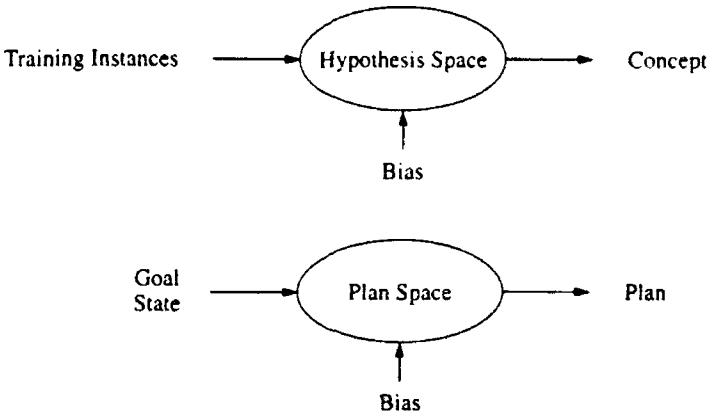


Figure 1. Analogy between concept learning and planning.

unbiased hypothesis space not expressible in the language. Because absolute biases introduce incompleteness into the process, devising a "good" bias is critical: if it is too weak it has no effect, but if it is too strong it can eliminate the desired output. A relative bias defines a partial order on the elements of the hypothesis space. Returning to the concept learning case, a preference for simpler hypotheses provides a relative bias. Relative biases do not generate incompleteness, but rules learned from relative biases tend to be more complex than those learned from absolute biases, so that the utility of learned rules may be decreased. In the case of planning, both absolute and relative biases have been used, though not generally under these labels.

On the absolute side, two common planning biases are *linearity* and *protection.* A linearity bias removes from the hypothesis space all plans in which operators in service of different unachieved goal conjuncts occur in succession; that is, once an operator for one unachieved goal conjunct is in the plan, operators for other conjuncts can occur only after the first goal conjunct has been achieved. For example, given the initial state and the goal conjuncts in Figure 2(a), plans such as the one in Figure 2(b) would be eliminated, while plans such as the one in Figure 2(c) would remain. A protection bias eliminates all plans in which an operator undoes a goal conjunct established by an earlier operator in the

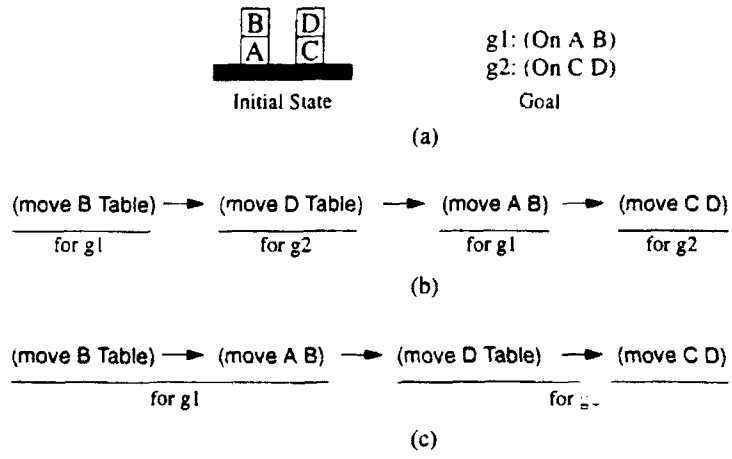


Figure 2. Example of the effects of a ***linearity*** bias on the plan space: (a) initial state and goal conjuncts, (b) plan eliminated, (c) plan remaining.

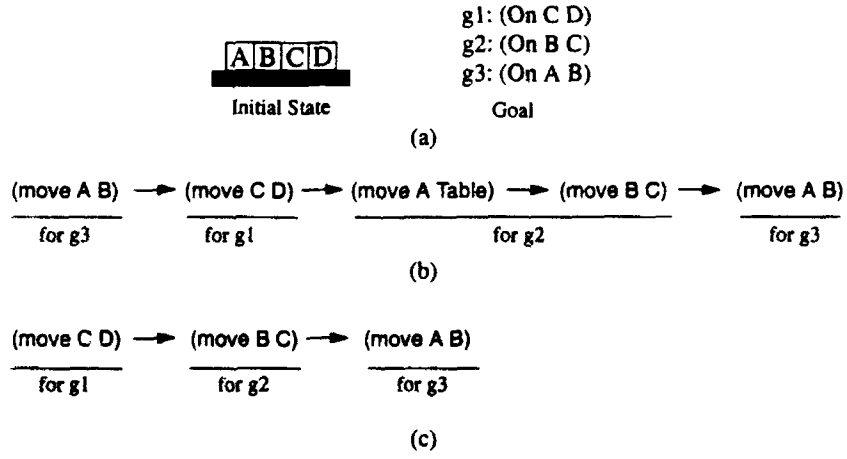


Figure 3. Example of the effects of a ***protection*** bias on the plan space: (a) initial state and goal conjuncts, (b) plan eliminated, (c) plan remaining.

sequence. For example, given the initial state and the goal conjuncts in Figure 3(a), plans such as the one in Figure 3(b) would be eliminated since the operator (move A Table) undoes the goal conjunct (On A B) which is established by the earlier operator (move A B), while plans such as the one in Figure 3(c) would remain.

Two less common, but nonetheless interesting, absolute biases are *directness* and *nonrecursiveness*. A directness bias eliminates all plans in which there is at least one operator that does not directly achieve a goal conjunct included in the problem definition. For example, given the goal conjuncts and operators in Figure 4(a), plans such as the one in Figure 4(b) would be eliminated since the operator (move B A) does not directly achieve any of the goal conjuncts in the problem definition, while plans such as the one in Figure 4(c) would remain. A nonrecursiveness bias eliminates all plans that require a derivation embodying recursive subgoals. For example, given the goal conjuncts and operators in Figure 5(a), plans such as the one in Figure 5(b) would be eliminated because it requires a derivation embodying a recursive subgoal — operator (move B Table) is chosen in service of conjunct (Clear C), but in making it applicable, a recursive Clear conjunct (Clear B) is generated (resulting in the selection of (move A D) as the first operator). On the other hand, plans such as the one in Figure 5(c) would remain.

Because these biases are absolute, they all engender incompleteness in the planner; that is, they reduce the number of plans that the planner can possibly generate for particular problems. This incompleteness can be used to speed up the planner. However, it only really helps if the bias is an appropriate one; otherwise, the effort expended in searching the biased space is wasted. Thus, in order to show that using one of these absolute biases is reasonable, some form of appropriate justification is needed. The most common form of justification is an *independence assumption*. Linearity and protection both depend on some form of independence assumption. For linearity, one assumes that while solving one goal conjunct, operators not in service of that conjunct need not be considered. For protection, one assumes that while solving one goal conjunct, operators that interact negatively with previous goal conjuncts need not be considered. Other justifications include *progress* and *boundedness*. A progress assumption — that it is always possible to move forward, and never required to move backward — underlies all greedy biases, of which

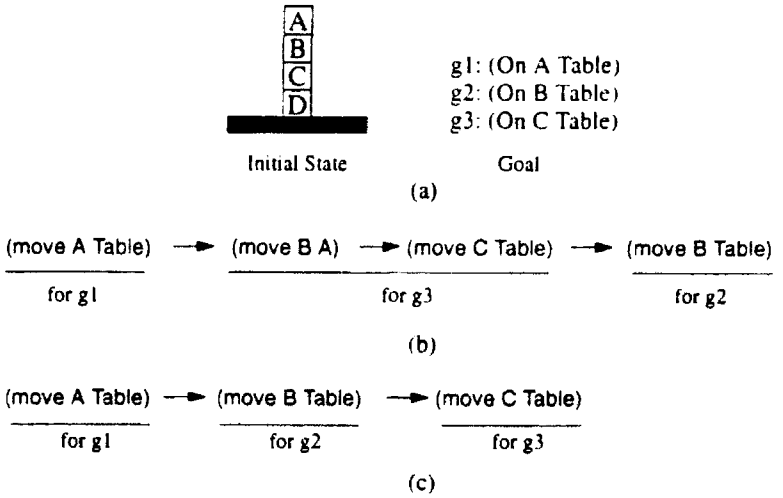


Figure 4. Example of the effects of a *directness* bias on the plan space: (a) initial state and goal conjuncts, (b) plan eliminated, (c) plan remaining.

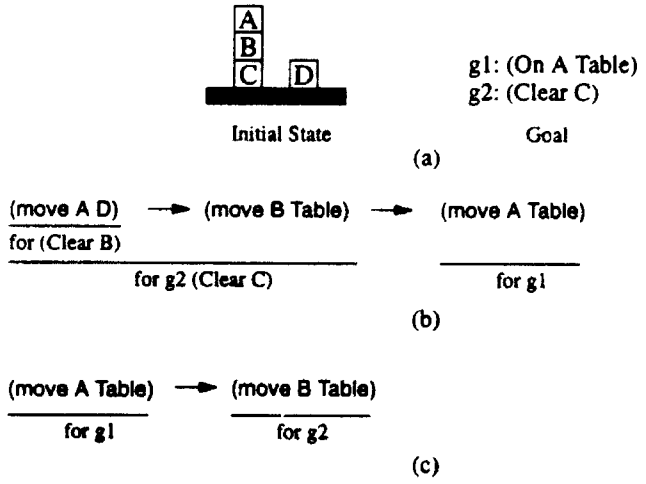


Figure 5. Example of the effects of a *nonrecursiveness* bias on the plan space: (a) initial state and goal conjuncts, (b) plan eliminated, (c) plan remaining.

protection is one. Boundedness assumptions limit the total effort that it is reasonable to expend in solving a problem. Nonrecursiveness and directness are both justified by boundedness assumptions, though based on different bounds.

Each of the absolute biases can also be made into a corresponding relative bias by just preferring plans that meet the bias to plans that do not. For example, a relative protection bias would prefer protected plans to unprotected ones, but still fall back on unprotected ones if necessary. In addition, there are a number of biases which are most naturally cast directly in relative terms. Three common relative biases are *abstraction*, *earliness*, and *shortness*. An abstraction bias prefers plans that can be generated by locally filling in the gaps in an abstract plan: an earliness bias prefers plans discovered earlier in the search; and a shortness bias prefers shorter plans. The primary justification for those biases is *efficiency*; specifically, planning efficiency for abstraction and earliness and execution efficiency for shortness.

The dichotomy between absolute and relative biases parallels the comparable dichotomy in the use of control knowledge. For example, Gratch and DeJong (1990) distinguish between *structural* and *ordering* modifications to control strategies, which amounts to a distinction between the addition of absolute and relative control knowledge. The two distinctions are also closely coupled, as imposition of a bias of one type can engender control of that same type, and vice versa.

LEARNING AND PLANNING IN SOAR

Our investigations of biased planning, and its influence on learning, have been performed in the context of Soar, an architecture that integrates basic capabilities for problem-solving, use of knowledge, learning, and perceptual-motor behavior (Laird, Newell, & Rosenbloom, 1987; Rosenbloom et al., 1991). Soar has not traditionally been seen as a planning architecture, partly because it does not create structures that resemble traditional plans, and partly because its problem-solving approach does not closely resemble the traditional planning methods. However, appearances can be deceiving. In this section we first very briefly review familiar territory, summarizing how learning works in Soar, and then ex-

amine planning in Soar from the perspectives of both plans and planning methods.

Learning in Soar

Soar learns via a chunking process that creates new rules that can recreate the results of subgoals in relevantly similar future situations (Laird, Rosenbloom, & Newell, 1986). For each independent result of each subgoal it creates a rule that has an action side based on the result, and a condition side based on a dependency analysis of the subgoal processing that led to the result. In effect, chunking is much like explanation-based learning (Rosenbloom & Laird, 1986).

The Soar representation of plans

This is not the place to attempt resolution of the philosophical questions over what is and is not a plan. However, enough of a working definition is needed to allow the identification of what structures in Soar act as plans. Thus, for the purpose of this identification, we will assume the following generic definition of a plan for a problem (that is, a state and a goal):

> A *plan* for a problem is a structure that represents the sequence of actions to be taken for that problem.

This definition captures a number of the important aspects of what it means for a structure to be a plan: that a plan is a representation of actions, that the actions in the plan have not yet taken place, and that a plan is for the solution of some problem (or class of problems). The definition is also neutral on a number of issues for which there is no present need to take a stance: the way the plan is encoded (whether declaratively or procedurally, with what syntax, and with what degree of expressibility), by whom the plan was created (the agent that is to execute it or some other other agent), and to whom the plan is representational (to the agent or to an external analyst). Other aspects ignored by this definition which may ultimately be of importance are: whether the structure is operational for control — for example, whether the plan can lead to

action in a bounded amount of time or whether something like exponential theorem proving is required to derive indirect action consequences — and whether there is a deliberate act of creation or selection of the plan for the problem.

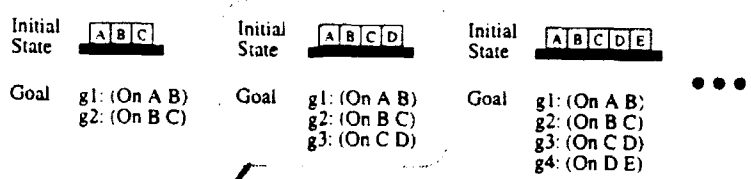
With this definition of a plan in hand, it is now possible to identify the two predominant components that serve as plans in Soar: (1) sets of instantiated preferences in preference memory[4] serve as instantiated plans for active goals; and (2) sets of variabilized control rules in production memory serve as generalized plans for classes of potential goals. To illustrate this, Figure 6(a) contains a generalized plan for the class of block-stacking problems shown in Figure 6(b). This single-rule generalized plan gets instantiated once for each successive triple of blocks in a desired stack, avoiding the mistake of putting the top block on the second block until it is itself already in place on the third block. Figure 6(c) shows the sequence of steps for a four-block-stacking problem. For each step it shows the problem state, the conjuncts that have not yet been achieved, the operators that have been proposed, and the portion of the instantiated plan — that is, the set of *worst* preferences — that applies at that step. Figure 6(d) shows the actual operator sequence this plan generates.

The reason that Soar does not appear to have plans is that they are rarely represented as unitary entities. The generalized plan in Figure 6(a) consists simply of the set — in this case a singleton set — of control rules, out of what would be the entire set of rules in memory, that are relevant to the class of problems in Figure 6(b). For other classes of problems, some of these same rules may be relevant, while others may not be. Likewise, the instantiated plan in Figure 6(c) cuts an unnatural swath through Soar's preference memory. First, it ignores the preferences that might simultaneously be in preference memory for other goals. Second, it contains preferences for an entire sequence of decisions, whereas preference memory focuses on preferences for the currently active decisions; that is, it would only contain the preference subset for one step at a time. The instantiated plan is thus assembled dynamically, and through time, rather than existing as a static unitary structure that can easily be read

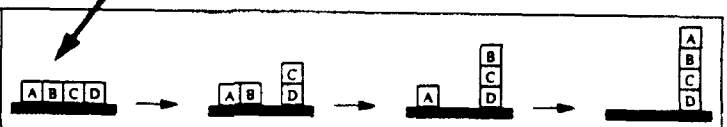
4. In contrast to previous versions, in Soar5 — the current major release — transiently retrieved preferences are maintained in a separate preference memory, rather than in working memory (Laird et al., 1990).

Goal protection can hold
Want a stack of at least three blocks
Neither of the top two blocks (out of the three) are in position
Both of the top two blocks (out of the three) are clear
An operator is proposed to put the top one on the second one
-->
The operator is worst

(a)



(b)

State			
Unachieved Goal Conjuncts	(On A B) (On B C) (On C D)	(On A B) (On B C)	(On A B)
Proposed Operators	(move A B) (move B C) (move C D)	(move A B) (move B C)	(move A B)
Instantiated Plan	(move A B) is worst (move B C) is worst	(move A B) is worst	

(c)

(move C D) → (move B C) → (move A B)

(d)

Figure 6. Goals, plans, and operator sequences: (a) a generalized plan, (b) the class of problems for (a), (c) the sequence of steps for a four-block-stacking problem, (d) the sequence of operators.

off by an external observer.

The generalized plan representation combines aspects of three existing formalisms — linear operator sequences, partially-ordered operator sequences, and stimulus-response rules — but it also goes beyond them in several ways. The preference language, in common with linear operator sequences and stimulus-response rules, has an imperative construct (*best*) that allows relatively direct specification of the next action to perform; however, it also goes beyond this to allow, in common with partially-ordered operator sequences, specification of partial order — using binary preferences such as *worse* and *better* — as well as beyond this to operator avoidance (*worst* and *reject*). The use of control rules, in common with stimulus-response rules, provides a fine-grained conditionality and context sensitivity that allows it to easily encode such control structures as conditionals and loops. In addition, the variabilization of the control rules allows a single plan fragment to be instantiated for multiple related decisions.

Planning methods in Soar

At the problem-solving level, Soar is based on the idea of multiple problem spaces (Newel et al., 1991) — that is, on multiple sets of operators and states, their selection, and the application of operators to states to yield new states — and their interaction through goal-subgoal interfaces. Early work on Soar demonstrated how this organization, in conjunction with small amounts of additional knowledge — structured as *method increments* — could yield a wide range of standard problem-solving methods. Although this included means-ends analysis (MEA) — the primordial planning method — most of the exhibited methods, such as depth-first search and hill-climbing, did not resemble classical planning methods. Thus Soar was conventionally viewed as a search system rather than as a planning system. However, recent work on a Soar-based framework for planning has demonstrated how versions of such standard planning methods as linear, nonlinear,[5] and abstraction (hierarchical)

5. The term "nonlinear" has several different meanings in the world of plans. The specific sense intended here is that operators generated in service of different goal conjuncts can be interleaved.

planning can be derived by adding method increments that include core means-ends knowledge about what operators to suggest for consideration, and varying knowledge about how to respond to impasses resulting from precondition failures (Rosenbloom, Lee, & Unruh, 1990).

Figure 7 provides initial traces of how particular versions of these three forms of planning proceed, in their current Soar implementation, for Sussman's anomaly (in the blocks world).[6] They all start with a high-level operator that is to achieve the entire conjunctive goal — (On B C) and (On A B) — directly from the initial state, and reach an execution impasse if there is no information about how to do this. In response to this impasse, a subgoal is created where means-ends analysis is used to generate the set of candidate operators — (move B C) and (move A B) — that may be able to achieve any of the goal conjuncts. A selection impasse then occurs unless there is information about how to pick among them (or unless only one operator is generated). In this selection impasse, the alternatives are evaluated by simulating their consequences. The simulation begins by selecting one of the alternatives to evaluate — here it is (move A B). Its preconditions are tested and if it is known to be applicable, it is executed. If it is not known to be applicable, what happens next depends on whether or not there is abstraction. With abstraction, the operator is executed anyway and problem solving just continues. In Figure 7(c), for example, operator (move A B) is executed even though block A is not clear. Without abstraction, as in Figure 7(a) and (b), an execution impasse occurs again. In response to this impasse, a new set of goal conjuncts is generated from the operator's unmet preconditions.

The difference between linear and nonlinear planning, at least for these versions, is in how the focus of operator generation shifts from the original set to a set including these new ones. Linear planning follows a stack discipline, where attention shifts completely to these new conjuncts — (Clear A) in this example — stays with them until they are achieved, and then pops back to the original conjunct that led to the impasse. Once the original conjunct is achieved, processing shifts to one of its siblings (if there are any). Nonlinear planning instead shifts to an expanded set

6. For comparison purpose, we are showing abstraction in the blocks world. Although we have not actually implemented it in that domain, it has been implemented in several similar domains.

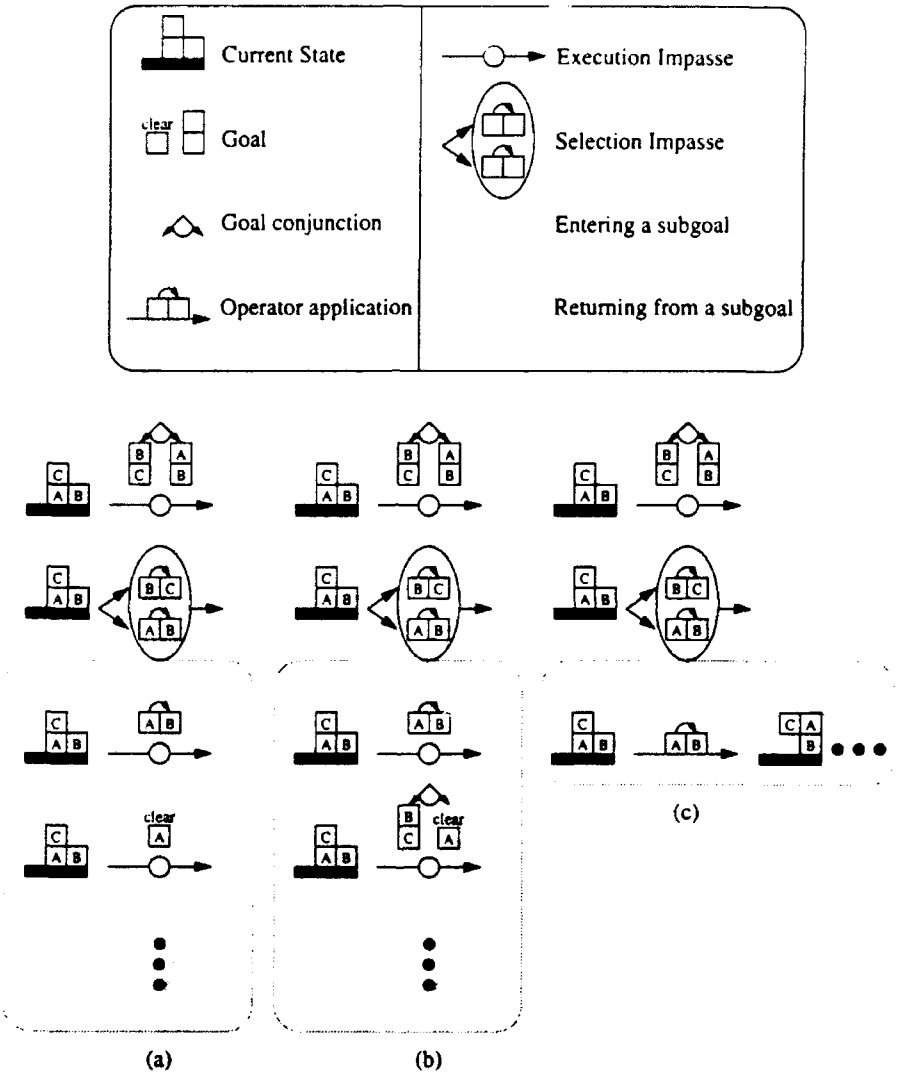


Figure 7. Planning in the blocks world using (a) linear; (b) nonlinear; and (c) abstraction (hierarchical) biases.

of conjuncts that includes the new set plus the original set minus the conjunct that led to the impasse, yielding (Clear A) and (On B C) in this example. At any point in time, an operator can be selected for any of these conjuncts, enabling operator sequences to be interleaved as necessary (similar to the *casual-commitment* approach to nonlinear planning (Veloso, 1989)). For both planning methods, once the new focus has been determined, planning continues recursively by using means-ends analysis to generate candidate operators for the new set of goal conjuncts.

Although we have so far been referring to these methods as "planning methods", because they are versions of classical methods used in the creation of plans, nothing has yet been said about how they in fact yield plans — that is, sets of either instantiated preferences or generalized control rules. Plans — actually *plan fragments* — are generated whenever operator preferences are created in working memory. This can happen simply by the instantiation of a generalized plan fragment — that is, by the execution of a control rule — or by the returning of a result from an operator-selection subgoal. For example, in Figure 7(a) a best preference is returned from the selection subgoal if the result of evaluating (move A B) is success, whereas a worst preference is returned if the result is failure. These preferences act directly as fragments of a plan for the currently active goals. In addition, whenever a preference is returned as a result of a subgoal, it triggers Soar's chunking process, which creates and stores a control rule that acts as a generalized plan fragment for classes of problems.

Most plan fragments that are not created simply by instantiating generalized plans are generated from projection (that is, lookahead) episodes in subgoals. In projection, one or more domain operators are tried out in simulation to see which ones lead to success or failure. Success engenders *best* preferences and failure engenders *worst* preferences. Thus projection plays an integral role in determining which plans are created. In its turn, what is projected, and what is considered to be success or failure, is determined by the planning method. These relationships are summarized by the following two influence paths.

planning method $\succ$ projection $\succ$ instantiated plan

planning method $\succ$ projection $\succ$ learning $\succ$ generalized plan

Within this framework, planning biases are implemented by altering the planning method, which then — through the influence paths above — determines which plans are created. For example, a protection bias is implemented by altering the planning method to terminate lookahead with failure any time a projected path leads to a protection violation. In comparison to the same planner without this bias, the protection planner will lead to the creation of worst preferences (and negative control rules) which will avoid paths that violate protection. If relative biases are used, it should also be possible to learn control rules that generate binary preferences (such as *better* and *worse*) encoding partial-order information, but this has not yet been investigated (at least in the context of bias and planning).

IMPLEMENTED PLANNING BIASES

The planning biases that we have concentrated on recently are linearity, protection, directness, and abstraction. The first three have all been implemented as options within a single planning system — and will be the focus here — while the latter has been implemented separately (Unruh & Rosenbloom, 1989). The implemented system that combines linearity, protection, and directness is constructed as a nonlinear planner that can optionally employ any of several different bias values along two independent dimensions — *goal flexibility* and *goal protection*. The nonlinear planner is described in Figure 7(b). It uses means-ends analysis on the entire set of goal conjuncts to decide which operators to consider for selection, performs search to decide among the set of considered operators, and generates new goal conjuncts whenever one or more preconditions of the selected operator are not achieved.

The goal-flexibility dimension is shown in Figure 8. It ranges over the planner's degree of flexibility in the pursuit of subgoals for precondition failures, and subsumes the directness and linearity biases. The most restricted point along this spectrum disallows all pursuit of subgoals for precondition failures (Figure 8(a)), yielding a single-level subgoal hierarchy. That is, if an operator has unsatisfied preconditions when the attempt is made to incorporate it into a plan, that plan — actually, any plan incorporating that operator in that role — is rejected. This

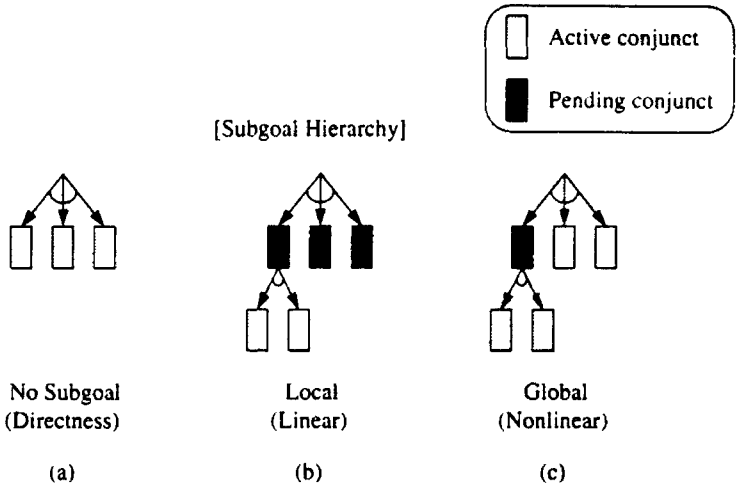


Figure 8. The dimension of goal-flexibility bias.

implements a directness bias because means-ends analysis ensures that operators are only considered if they achieve a goal conjunct,[7] and the only goal conjuncts that are allowed are the ones in the initial problem specification, so no operators are allowed in the plan except for those that directly achieve goal conjuncts in the initial problem specification.

The second point along the flexibility dimension allows the local use of subgoals (Figure 8(b)). Here, precondition failures lead to generation of new goal conjuncts, but only a single local set of conjuncts are attended to at any point in time. Initially the local set consists of the conjuncts in the problem specification. However, whenever a selected operator has one or more unmet preconditions, the previous local set is pushed on a stack, and the operator's unmet preconditions become the new local set. When the operator's conditions are satisfied, the stack is popped to return to the previous set. This local focus of attention has two main consequences for the planner. First, it reduces the branching factor

7. Means-ends analysis in general may not guarantee this, but the restricted form of MEA typically used by planning systems — where it is based on the unification of operator actions with goal conjuncts — does guarantee it.

of the planner's search — with respect to the nonlinear planner — by restricting the set of operators that the planner can consider at any point in time to just those that may achieve the local conjuncts. Second, it enforces linearity on the resulting plans by restricting the placement of an operator to within the context of the local conjuncts from which it arose.

The third point along the flexibility dimension allows the global use of subgoals; that is, new goal conjuncts are generated for unmet preconditions, and operators are simultaneously considered for all unsatisfied conjuncts (Figure 8(c)). This is the least restricted version, and enables nonlinear planning by allowing operators for multiple goal conjuncts to be interleaved.

The two points implemented along the goal-protection dimension correspond to full goal protection — that is, no achieved goal conjunct in the plan can be violated — and no goal protection. The main consequence of imposing full goal protection is that the search tree is reduced in size because paths that violate goal protection are cut off before full plans are created.

Figure 9 characterizes a 3×2 set of planning methods derived from these bias dimensions. The most biased planner (P1) is at the top-left corner of the figure. This is a direct goal-protection planner. Although quite restrictive, it is sufficient to solve the block-stacking problem shown in that cell of the figure. The least biased planner (P6) is in the bottom-right corner of the figure. It is a nonlinear planner without goal protection, and is the only planner in the figure capable of generating an optimal solution to the blocks world problem shown in that cell.[8] Between these two extremes, moving up or to the left yields more bias, while moving down or to the right yields less bias. In each of these intermediate cells, the problem shown is one that is just hard enough to require that planner; that is, the problem can be solved optimally by the planner represented by that cell, but not by either the planner to its left or the planner above it.

Tables 1(a-c) show the same 3×2 matrix, but each cell now contains

8. In this domain both P5 and P6 are complete planners in that they can potentially solve every problem (though P5 may not be able to generate an optimal solution). However, in domains with irreversible operators, P6 is the only complete planner.

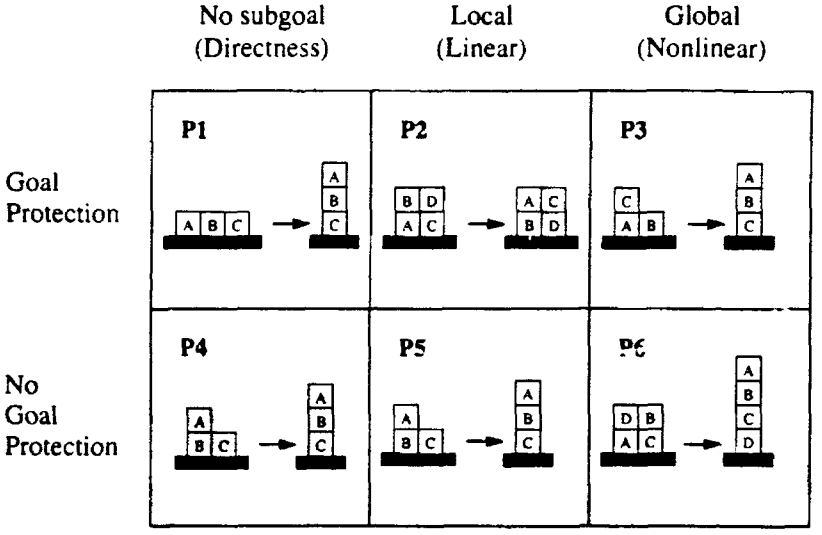


Figure 9. The planning methods generated by the bias dimensions (the bottom-left corner represents an extended blocks world problem where a block that is not clear can be moved, dropping all the blocks above it onto the table).

experimental results bearing on the trade-offs between efficiency and completeness for these six planners. These data come from running each planner on three replications of the same set of fifty blocks-world problems, for a total of 150 trials each. Problems are repeated to help average out the variations caused by nondeterminism in the planners — whenever they reach a decision at which they are indifferent among the set of alternatives, one alternative is picked at random. The first ten problems out of the fifty all involve two blocks and two goal conjuncts. For each ten subsequent problems the maximum number of blocks and goal conjuncts were each increased by one (the last ten problems thus have a maximum of six each). For each problem, an initial state was randomly generated containing between two and the (current) maximum number of blocks. Likewise a set of goal conjuncts was randomly generated that numbered between two and the number of blocks in the initial state.

Learning was turned on for each problem, but only within-trial transfer was allowed; that is, rules learned during one problem were not used for other problems. This provides a generalized form of dependency-directed backtracking, but does not get into the issues of across-problem interactions.

Table 1(a) shows the number of problems solvable in principle — that is, if sufficient time is provided — by that cell's planner, plus a label for the problem set that this implicitly defines. Not surprisingly, this shows a monotonic trend between planner bias and scope, from a low of 28 problems for the most restricted planner to a high of 50 problems for the least restricted planner. Tables 1(b) and (c) show the average number of decisions and the average plan length, which should positively correlate, respectively, with planning time and execution time. This data arises from applying each of the six planners to those of the four problem sets defined in Table 1(a) that they can in principle solve. The four problem sets are associated with the four rows within each cell of the tables. The averages in each cell only include the data from the trials that were solved within an a priori limit of 300 decisions. Since 99% of the solvable problems were actually solved within this limit, this includes nearly all of the trials.

The timing results in Table 1(b) show that planning effort is a monotonically decreasing function of the amount of bias along these dimensions. For example, for problem set S1, effort ranged from a low of 16.5 decisions for the no-subgoal (direct) methods to a high of 36.6 decisions for global flexibility (nonlinear planning) without protection. If this trend holds more broadly across other domains, the resulting trade off between efficiency and completeness — efficiency decreases as the bias is relaxed, while completeness increases — implies that context-sensitive bias selection will be critical for finding solutions quickly across broad ranges of problem difficulty.

Plan length in Table 1(c) shows a similar monotonic trend, though there is one reversal when going from linear to nonlinear planning (both without goal protection). The most likely cause of this reversal is that the linear planner's bias is weak enough to allow solutions to be found for all blocks world problems, but strong enough to eliminate optimal solutions for some of the problems; for example, when the shortest plan

No subgoal	Local	Global (Directness)	(Linear)	(Nonlinear)
Goal Protection		28 (S1)	45 (S2)	46 (S3)
No Goal Protection		28 (S1)	50 (S4)	50 (S4)

(a) Number of problems solvable in principle.

		No subgoal (Directness)	Local (Linear)	Global (Nonlinear)
Goal Protection	S1	16.5	17.2	17.5
	S2	-	28.9	36.4
	S3	-	-	36.4
	S4	-	-	-
No Goal Protection	S1	16.5	23.2	36.6
	S2	-	38.8	50.7
	S3	-	43.4	54.5
	S4	-	45.1	58.0

(b) Average number of decisions per problem solved.

		No subgoal (Directness)	Local (Linear)	Global (Nonlinear)
Goal Protection	S1	1.7	1.8	1.8
	S2	-	2.8	3.0
	S3	-	-	3.0
	S4	-	-	-
No Goal Protection	S1	1.7	2.3	2.5
	S2	-	3.7	3.6
	S3	-	4.1	4.0
	S4	-	4.3	4.1

(c) Average plan length per problem solved.

Table 1. **Results from the six planners on three independent repetitions of fifty randomly generated blocks world problems.**

requires operators in service of different goal conjuncts to be considered before the current goal conjunct is achieved.

MULTI-METHOD PLANNERS

The ideal planner would be able to solve each problem with a minimum of excess work (at both planning and execution time). However, each of the planners examined in the previous section is either incomplete or performs a significant amount of excess work for some of the problems. Given this, an alternative way to approach the ideal is to construct a multi-method planner that uses, for each problem, the least costly planner that is sufficient for it. Toward this end we have created a set of multi-method planners, each consisting of a sequence of primitive planners. For each multi-method planner, planning starts with the most restricted planner, and falls back on failure to successively more relaxed planners until one is found that is sufficient for the problem. The general approach is similar to how multiple biases have been used in inductive concept formation (Rendell, 1986; Russell & Grosof, 1987; Utgoff, 1986), how preservable constraints are relaxed on failure in FAILSAFE-2 (Bhatnagar & Mostow, 1990), and how rejection (absolute) biases are weakened in (M. Barley, personal communication, 1991); however, there are a number of differences in the details, such as which biases are used, how it is decided to weaken the biases, how much the biases are weakened at one time, etc.

Multi-method planners are implemented via a bias space containing operators that set the bias. Without knowledge about which biases work for a problem, a lookahead search is performed in which the biases are tried out in the sequence specified — in essence the system is projecting on the bias as well as on the domain operators. As soon as a bias is found that works for the problem, the lookahead search is terminated, and that bias is applied in actually solving the problem. Figure 10 shows a trace of multi-method planning for a sequence of three planners: (1) the most restricted planner, P1 (direct goal-protection); (2) an intermediate planner, P3 (nonlinear goal-protection); and (3) the least restricted planner, P6 (nonlinear no-protection). The planner starts by evaluating the planning methods. In the case of Sussman's anomaly, as shown in

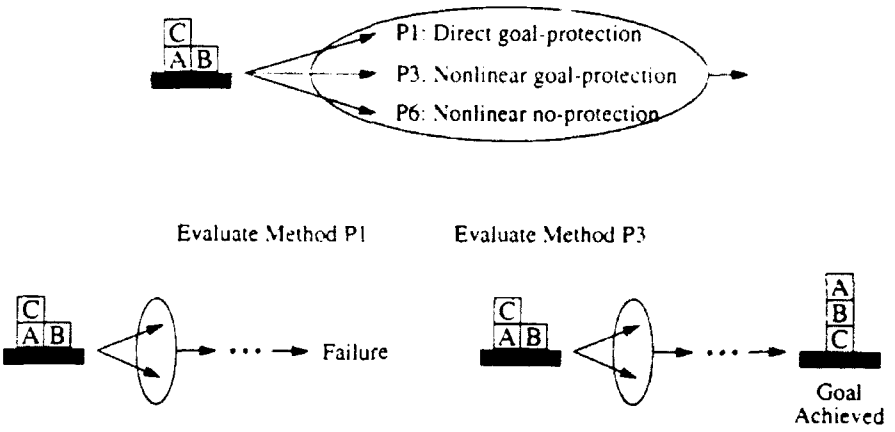


Figure 10. A trace of multi-method planning.

the example, the evaluation of the direct goal-protection method returns failure because the goal conjunct (On A B) cannot be achieved without generating a new subgoal. Once a method fails, the next most relaxed method — in this example it is the nonlinear goal-protection method — is tried, and so on, until a solution is found.

Table 2 compares the performance of the two single-method planners that are complete for this domain — the linear no-protection planner (P5) and the nonlinear no-protection planner (P6) — versus four multi-method planners. Since all of the multi-method planners in this table contain a complete single-method planner, they are also complete. For each of these six complete planners, the table shows the sequence of primitive planners out of which it is composed, the average number of problems that it actually solved within 300 decisions (averaged over the same set of 150 trials as in the previous section), the average number of decisions for the solved problems, and the average plan length for the solved problems. To aid in comparing the methods, the parenthetical numbers in the last two columns provide the same data for the 43 problems that all six methods solved on all three repetitions. The results on

Planning type		Average number of problems solved	Average number of decisions	Average plan length
Single-method Planning	P5	48.7	45.1 (32.3)	4.3 (3.3)
	P6	48.7	58.0 (44.6)	4.1 (3.2)
Multi-method Planning	P1 - P5	48.3	44.3 (37.9)	3.5 (2.8)
	P1 - P6	48.7	58.5 (46.4)	3.8 (3.0)
	P1 - P2 - P5	46.7	43.3 (40.7	3.0 (2.8)
	P1 - P3 - P6	47.7	53.3 (46.1)	3.1 (2.7)

Table 2. Single-method versus multi-method planning.

these common problems reveal a monotonic trend whereby adding more planners marginally increases planning time, in exchange for reductions in plan length. In general the linear planners were quicker than the nonlinear planners, while there was no strong pattern for plan lengths.

Ideally the multi-method planners would not only have produced shorter plans, but would also have taken less time to do so. The idea is that problems solvable by more restricted planners should be solved more quickly, while problems requiring less restricted planners should not waste too much extra time trying out the insufficient early planners. The intuition behind this is based on iterative deepening (Korf, 1985). In iterative deepening, a sequence of depth-first searches are performed, each to a greater depth than the previous one. If a solution is found at a shallow depth, the cost of searching to a greater depth is saved. If a solution is not found at a particular depth, a deeper search is performed. The cost of doing the shallower searches is then wasted, but since the deeper search costs at least B times the cost of the shallower search — where B is the branching factor of the search tree — this cost can be relatively quite small. Thus, if the proportion of problems solvable at shallow depths is large enough, and the ratio of costs for successive levels is large enough, there should be a net gain. However, the results in Table 2 show that, at least for these methods and problems, these assumptions are not met. The planning-time balance is instead in favor of the single-method approaches.

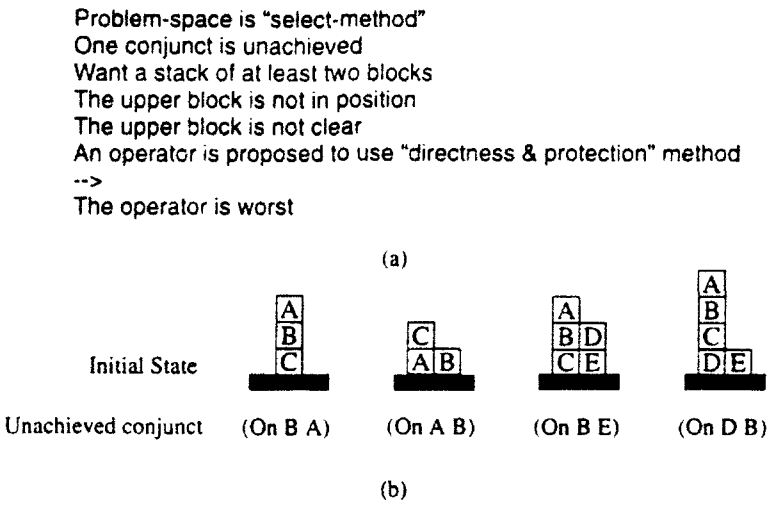


Figure 11. Example of learning which planners to use for which classes of problems: (a) a learned rule to avoid the direct goal-protection planner, (b) a class of problems in which this rule is applicable.

One way to further ameliorate the effects of wasting effort on insufficient planners is to use learning, in particular of two sorts. The first sort of learning is about which planners to use for which classes of problems. To the extent that this can be done, the effort wasted in trying inadequate methods can be avoided. In our Soar-based implementation, bias selection is structured just as would be any other selection, so this sort of learning can happen automatically by chunking. From an experiment with such learning, Figure 11 shows a rule learned to avoid using the most restricted method — that is, direct goal-protection — under specific circumstances where there is only one goal conjunct but (at least) two blocks must be moved to achieve it. This rule was learned during the first problem and used in three later problems to avoid even trying this method.

The second sort of learning is within-planner learning that can transfer across planners (possibly for the same problem). If a projection is performed within one planner, and the results of the projection depend

only on aspects of the planner that are shared by a second planner, then it should not be necessary to repeat that projection when the second planner is tried. For example, the rule in Figure 6(a) is learned from a plan violating goal protection in the direct goal-protection planner and transfers to the nonlinear goal-protection planner, where it prevents the planner from reprojecting along paths that violate goal protection.

Though we have examined instances of both of these forms of learning in the context of multi-method planning, no systematic study has yet been made of their effectiveness or of whether issues of overgeneralization and/or undergeneralization will prove troublesome. Future work should include rerunning the experiments summarized in Table 2 with both of these forms of learning enabled.

Another potential way to improve the performance of multi-method planners is to reduce the granularity at which the individual planning methods are selected and used. If there are a significant number of problems where most of the subgoals are solvable by a very cheap method (such as directness) while the remainder of the problem requires a more complex method (such as linear planning), then making an independent bias selection each time the planner recurs on a set of subgoals may allow focused reductions in planning time and plan length. There would be increased overhead because of the extra decisions, but that may be more than compensated for by the use of simpler methods. This is all quite speculative for now, but does provide an interesting future area for investigation.

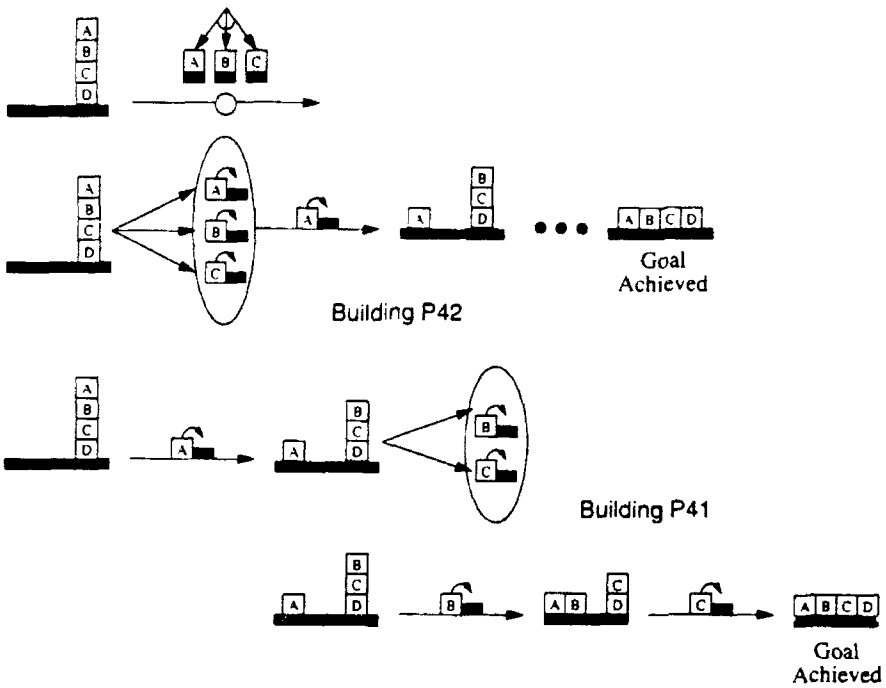
BIAS IN EXPLANATION-BASED LEARNING

As defined in the introduction, the bias in concept learning is anything other than strict consistency with the observed training instances that influences which generalization is chosen. Thus, when an explanation-based method is used for concept learning, its bias includes the entire set of inputs exclusive of the training example — that is, the domain theory, the goal concept, and the operationality criterion (Mitchell, Keller, & Kedar-Cabelli, 1986) — plus any other factors that influence which explanation is used and which definition/rule is extracted from the explanation. Each of these factors has been varied in at least one recent research

effort in service of increasing the utility of the rules acquired by EBL: restrictions on domain theory expressiveness (Tambe, Newell, & Rosenbloom, 1990);[9] variations in target (goal) concepts for acquiring control knowledge (Minton et al., 1989); variations in the operationality criterion (Braverman & Russell, 1988; Letovsky, 1990; Segre, 1987); explanation selection based on criteria such as coverage or (non)recursiveness (Cohen, 1990; Etzioni, 1990); and postprocessing via a range of deductive and inductive transformations (Chase et al., 1989; Cohen, 1990; Flann & Dietterich, 1989; Minton, 1988; Shavlik, 1989).

In the framework of this chapter, explanation-based learning of plans is performed over the planner's projection process: the elements to be explained are the preferences generated during projection, and the explanations are the traces of the projections that led to the preferences. Thus, if the planner's bias is reflected in an altered planning method, which in turn yields an altered projector, then the planner's bias can indirectly induce a bias in the resulting EBL process. Directness provides a simple example of this. Figure 12(a) shows a path projected without directness, by the nonlinear planner, for a simple four-block-unstacking problem. This projection proceeds through multiple selection impasses until the problem is successfully solved. As shown in Figure 12(b), this results in a pair of positive control rules, one for each correct decision on the solution path. These rules are relatively specialized, because each must encapsulate the entire explanation for why a particular operator will eventually lead to success. In larger problems these explanations get even larger, and the rules end up being even more specialized. Figure 13(a) shows a path projected with directness, for the same block-unstacking problem. In contrast to the previous case, this projection is terminated with failure as soon as the non-applicable operator (move B Table) is selected. The explanation for this failure is quite short — based as it is on the explicit assumption that directness can hold and on the failure of the first selected operator to be applicable — yielding the negative control rule in Figure 13(b). As it turns out, this single rule is general enough to handle the entire problem, by removing from

9. The use of "low belief" or "overgeneral" domain theories for knowledge level learning is another form of domain theory variation that provides a qualitative improvement in utility — from symbol level learning to knowledge level learning (Flann & Dietterich, 1989; Rosenbloom & Aasman, 1990).



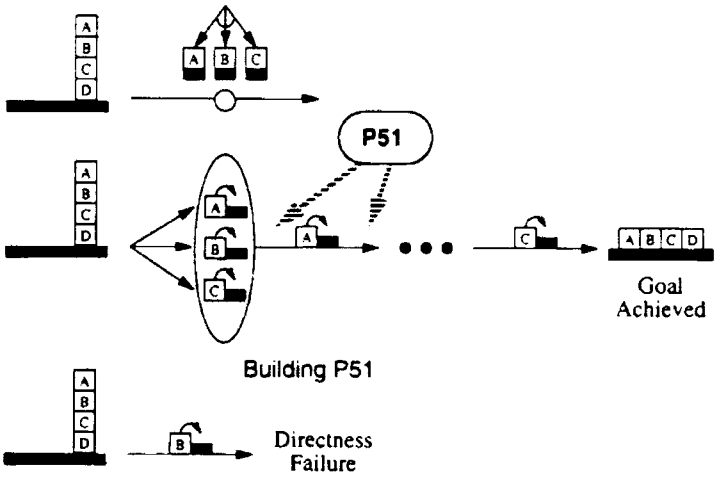
(a)

P41: Want two blocks on the table
One block is on top of the other
The top block is clear
An operator is proposed to put the top one on the table
-->
The operator is best

P42: Want three blocks on the table
These blocks are stacked one on top of another
The top block is clear
An operator is proposed to put the top one on the table
-->
The operator is best

(b)

Figure 12. Four block unstacking with nonlinear planning: (a) a projected path, (b) learned rules.



(a)

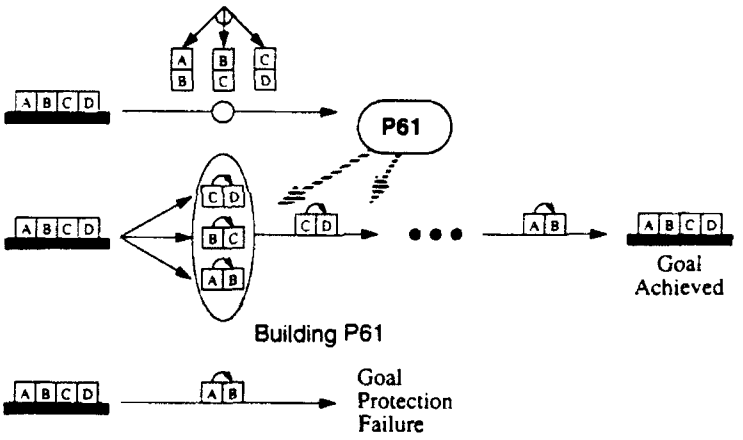
P51: Directness can hold
Want a block on the table
That block is not on the table
That block is not clear
An operator is proposed to put that one on the table
-->
The operator is worst

(b)

Figure 13. Four block unstacking with directness: (a) a projected path, (b) a learned rule.

consideration all operators that attempt to move unclear blocks onto the table. The bias in this case has thus yielded faster planning and learning — because of shorter projections and explanations — and has resulted in the acquisition of fewer, more general rules.

- Implicit in this example is one approach to producing *generalization to N* (Boström, 1990; Cohen, 1988; Shavlik, 1989; Subramanian & Feldman, 1990), where a plan learned for a problem of a particular size can transfer to solve problems with the same structure but of arbitrary size.



(a)

P61: Goal protection can hold
Want a stack of at least three blocks
Neither of the top two blocks (out of the three) are in position
Both of the top two blocks (out of the three) are clear
An operator is proposed to put the top one on the second one
-->
The operator is worst

(b)

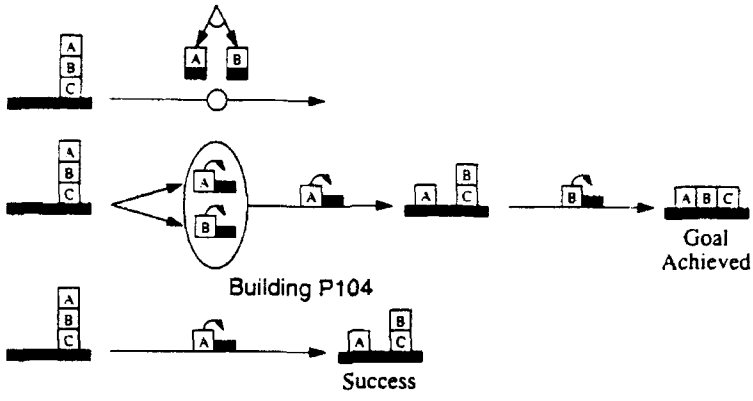
Figure 14. Four block stacking with protection: (a) a projected path, (b) a learned rule.

Without directness, the control rules are specific to particular numbers of blocks, and thus can only be used to directly solve terminal subregions of larger problems. However, with directness, a single rule is learned that removes from consideration at each decision all operators that move unclear blocks to the table, *no matter how many unclear blocks there are.* This idea can be applied to other problems and biases as well. Figure 14(a), for example, shows a path projected with protection for a four-block-stacking problem. As with the directness bias in block unstacking, a protection bias leads here to learning a single negative rule (Figure 14(b)) that can be applied to stacking problems of arbitrary size.

A third type of bias that can also induce generalization to N is *complete* protection. Complete protection is a variant on goal protection that provides a very strong bias by not only protecting established goals, but also protecting established operator sequences. That is, it disallows any backtracking on operator selection, thus letting projection be terminated with success whenever an operator is selected, rather than waiting until the entire problem has been solved. As with the directness example, projection is terminated here after the first operator is selected (Figure 15(a)). However, in this case it is terminated with success as soon as the top block is moved to the table. The explanation for this success depends only on the explicit assumption of complete protection and on the fact that the operator was successfully applied, so a relatively general, positive control rule is learned (Figure 15(b)). Although this is a positive rule, it also turns out to produce generalization to N, but now by always specifying that the one clear block that is not already on the table — if it were already on the table, there would be no active goal conjunct for it — should be moved to the table. The resulting rule can transfer to any number of iterations, as shown in Figure 15(c).

The key to producing generalization to N with these biases is that they enable learning from non-iterative paths — in this way it is similar to Etzioni's (1990) work on restricting EBL to learn from only non-recursive paths. In the directness and protection cases, the success paths are iterative, but (negative) rules can instead be learned from non-iterative failure paths. In the complete-protection case, learning occurs from a fragment of the success path that corresponds to just a single cycle of iteration. In both cases, the resulting rules can transfer to any number of iterations.

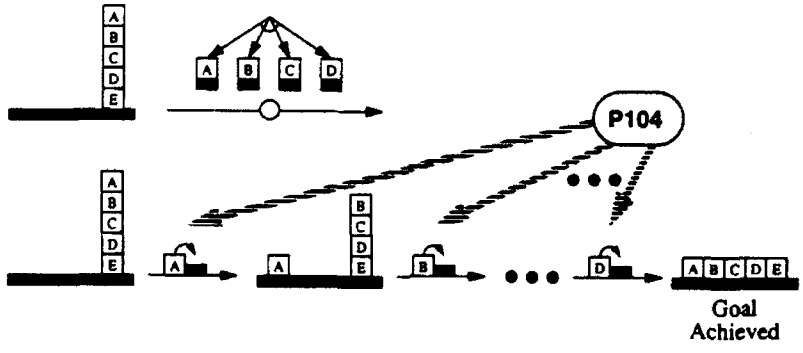
An even closer relationship to Etzioni's work could potentially be achieved by adding a nonrecursiveness bias to the planner. If this were added as an absolute bias, it would terminate projection with failure along any path that recurred. This would restrict learning to nonrecursive portions of projections, but would also go seriously beyond this to eliminate recursive plans from the space. Etzioni dealt with this by distinguishing those projections used for learning from those used for planning, and then only using the bias during learning. An alternative approach is to weaken nonrecursiveness by making it a relative bias. If all of the nonrecursive paths are projected without yielding enough pref-



(a)

P104: Complete protection holds
Want a block on the table
That block is not on the table
That block is clear
Operator is proposed to put that one on the table
-->
The operator is best

(b)



(c)

Figure 15. Four block unstacking with complete protection: (a) a projected path, (b) a learned rule, (c) transfer of the learned rule.

erences to generate an unambiguous plan, then it should still be possible to go back and project along the recursive paths. The result should be an overall preference for nonrecursive plans, and for recursive plans learned from nonrecursive projections, over recursive plans learned from recursive projections. Investigating such a relative nonrecursiveness bias in the Soar-based planner is an interesting possibility for future work.

Another example of engendering a useful bias on EBL by biasing the planning can be found in the work on learning from abstract planning (Unruh & Rosenbloom, 1989; Knoblock, Minton, & Etzioni, 1991). In this work, projections are performed with abstracted operator definitions rather than with the full operator definitions. The resulting abstract projections tend to be shorter and simpler than would be the comparable projections with unabstracted operators; so when rules are learned from these projections, they also tend to be shorter and simpler, and thus more general.

Given the evidence that planning biases can induce interesting and useful biases in EBL, and that in so doing the biases can assist both planning and learning, it is an interesting question to ask whether any other approaches to biasing EBL — such as post-hoc rule modification — are needed. Although it is premature to answer this question at this point, it is worth noting that the ultimate answer will depend on the scope of learning biases that can be generated in this fashion, and whether achievement of these learning biases requires distorting the planner to such an extent that it cannot properly achieve its task.

CONCLUSION

In this chapter we have taken seriously the notion of bias as a means of characterizing variations among planners. We hypothesized that this would yield three benefits: (1) help organize and understand many of the concepts in planning; (2) reduce the computational requirements of planning; and (3) induce effective biases in learning. On the first benefit, though it has not yet led to a complete theory or taxonomy of planning methods, it has led to the development of several orthogonal bias dimensions which provide fragments of organization over the space of methods. Six planners — defined by the cross-product of two bias

dimensions — have been implemented in Soar as variations on a single core planner. Further work is required to identify the remaining biases that underlie effective planning methods, and to build a unified planner that can optionally use arbitrary subsets of them.

On the second benefit, initial experiments with the six planners suggest (mostly) monotonic trade-offs between completeness and efficiency as the bias dimensions are traversed from least to most restrictive. In an attempt to move off of this trade-off curve, a set of multi-method planners were constructed from sequences of increasingly less biased planners. When the correct planner is not known a priori, a search is performed starting at the most restricted planner until a sufficient one is found. An experiment comparing these multi-method planners with the two complete single-method planners in the blocks world provided mixed results: reduced plan length no longer needed to be sacrificed for completeness, but reductions in plan length were accompanied by increases in planning time. Two learning strategies were presented as potential ways of further reducing the planning time required by the multi-method planners: acquisition of control rules that transfer among planners, enabling searches with more restricted planners to assist search with less restricted ones; and acquisition of rules that help select appropriate planners. These possibilities need to be investigated further, in conjunction with a deeper understanding of the entire set of trade-offs and experimentation in more realistic task domains. In addition, the idea of allowing a new method to be selected whenever the planner recurs on a new set of subgoals also needs to be examined.

On the third benefit, the effects of changes in planning bias on bias in explanation-based learning were investigated with a case study in generalization to N. Depending on the exact biases used, control rules were learned that either: (1) did not provide generalization to N (for nonlinear planning), provided it by eliminating all but the correct iterative option (for goal protection and directness), or provided it directly by specifying the correct iterative option (for complete protection). Although this is encouraging, considerable future work is still needed in evaluating the impact of the full span of planning biases on learning, and evaluating whether this provides a sufficient set of biases on learning.

REFERENCES

Bennett, J. S., & Dietterich, T. G. (1986). *The test incorporation hypothesis and the weak methods* (Technical Report 86-30-1). Department of Computer Science, Oregon State University.

Bhatnagar, N., & Mostow, J. (1990). Adaptive search by explanation-based learning of heuristic censors. *Proceedings of the Eighth National Conference on Artificial Intelligence* (pp. 895-901). Boston, MA: MIT Press.

Boström, H. (1990). Generalizing the order of goals as an approach to generalizing numbers. *Proceedings of the Seventh International Conference on Machine Learning* (pp. 260-267). Austin, TX: Morgan Kaufmann.

Braverman, M. S., & Russell, S. J. (1988). Boundaries of operationality. *Proceedings of the Fifth International Conference on Machine Learning* (pp. 221-234). Ann Arbor, MI: Morgan Kaufmann.

Chase, M. P., Zweben, M., Piazza, R. L., Burger, J. D., Maglio, P. P., & Hirsh, H. (1989). Approximating learned search control knowledge. *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 218-220). Ithaca, NY: Morgan Kaufmann.

Cohen, W. W. (1988). Generalizing number and learning from multiple examples in explanation based learning. *Proceedings of the Fifth International Conference on Machine Learning* (pp. 256-269). Ann Arbor, MI: Morgan Kaufmann.

Cohen, W. W. (1990). Learning approximate control rules of high utility. *Proceedings of the Seventh International Conference on Machine Learning* (pp. 29-33). Austin, TX: Morgan Kaufmann.

Etzioni, O. (1990). Why Prodigy/EBL works. *Proceedings of the Eighth National Conference on Artificial Intelligence* (pp. 916-922). Boston, MA: MIT Press.

Flann, N. S., & Dietterich, T. G. (1989). A study of explanation-based methods for inductive learning. *Machine Learning, 4*, 187-226.

Gratch, J. M., & DeJong, G. F. (1990). A framework for evaluating search control strategies. *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling, and Control* (pp. 337-347). San

Diego, CA: Morgan Kaufmann.

Knoblock, C. A., Minton, S., & Etzioni, O. (1991). Integrating abstraction and explanation-based learning in PRODIGY. *Proceedings of the Ninth National Conference on Artificial Intelligence* (pp. 541-546). Anaheim, CA: MIT Press.

Korf, R. E. (1985). Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence, 27*, 97-109.

Laird, J. E., Congdon, C. B., Altmann, E., & Swedlow, K. R. (1990). *Soar user's manual: version 5.2* (Technical Report CMU-CS-90-179). School of Computer Science, Carnegie Mellon University.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence, 33*, 1-64.

Letovsky, S. (1990). Operationality criteria for recursive predicates. *Proceedings of the Eighth National Conference on Artificial Intelligence* (pp. 936-941). Boston, MA: MIT Press.

Minton, S. (1988). *Learning search control knowledge: An explanation-based approach.* Boston, MA: Kluwer Academic Publishers.

Minton, S. (1990). Quantitative results concerning the utility of explanation based learning. *Artificial Intelligence, 42*, 363-391.

Minton, S., Carbonell, J. G., Knoblock, C. A., Kuokka, D. R., Etzioni, O., & Gill, Y. (1989). Explanation-based learning: A problem solving perspective. *Artificial Intelligence, 40*, 63-118.

Mitchell, T. M. (1980). *The need for biases in learning generalizations* (Technical Report CBM-TR-117). Department of Computer Science, Rutgers University.

Mitchell, T. M., Keller, R. M., & Kedar-Cabelli, S. T. (1986). Explanation based generalization: A unifying view. *Machine Learning, 1*, 47-80.

Rendell, L. (1986). A general framework for induction and a study of selective induction. *Machine Learning, 1*, 177-226.

Rosenbloom, P. S., & Aasman, J. (1990). Knowledge level and inductive uses of chunking (EBL). *Proceedings of the Eighth National Conference on Artificial Intelligence* (pp. 821-827). Boston, MA: MIT Press.

Rosenbloom, P. S., & Laird, J. E. (1986). Mapping explanation-based generalization onto Soar. *Proceedings of the Fifth National Conference on Artificial Intelligence* (pp. 561-567). Philadelphia, PA: MIT Press.

Rosenbloom, P. S., Laird, J. E., & Newell, A. (in press). *The Soar papers: Research on integrated intelligence*. Cambridge, MA: MIT Press.

Rosenbloom, P. S., Laird, J. E., Newell, A., & McCarl, R. (1991). A preliminary analysis of the Soar architecture as a basis for general intelligence. *Artificial Intelligence, 47*, 289-325.

Rosenbloom, P. S., Lee, S., & Unruh, A. (1990). Responding to impasses in memory-driven behavior: A framework for planning. *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling, and Control* (pp. 181-191). San Diego, CA: Morgan Kaufmann.

Russell, S. J., & Grosof, B. N. (1987). A declarative approach to bias in concept learning. *Proceedings of the Sixth National Conference on Artificial Intelligence* (pp. 505-510). Seattle, WA: MIT Press.

Segre, A. M. (1987). On the operationality/generality trade-off in explanation based learning. *Proceedings of the Tenth International Joint Conference on Artificial Intelligence* (pp. 242-248). Milan, Italy: Morgan Kaufmann.

Shavlik, J. W. (1989). Acquiring recursive concepts with explanation-based learning. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 688-693). Detroit, MI: Morgan Kaufmann.

Subramanian, D., & Feldman, R. (1990). The utility of EBL in recursive domain theories. *Proceedings of the Eighth National Conference on Artificial Intelligence* (pp. 942-949). Boston, MA: MIT Press.

Tambe, M., Newell, A., & Rosenbloom, P. S. (1990). The problem of expensive chunks and its solution by restricting expressiveness. *Machine Learning, 5*, 299-348.

Unruh, A., & Rosenbloom, P. S. (1989). Abstraction in problem solving and learning. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 681-687). Detroit, MI: Morgan Kaufmann.

Utgoff, P. E. (1986). Shift of bias for inductive concept learning. In

R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach, Vol. II*. Los Altos, CA: Morgan Kaufmann.

Veloso, M. (1989). *Nonlinear problem solving using intelligent casual-commitment* (Technical Report CMU-CS-89-210). School of Computer Science, Carnegie Mellon University.